DL-BASED INDUSTRIAL INSPECTION (DEFECT SEGMENTATION)

Peter Pyun Ph.D.
Andrew Liu Ph.D.

NVIDIA

Relevant Links:

Defect Segmentation

Nvidia Industrial Inspection White Paper V2.0:

https://nvidia-gpugenius.highspot.com/viewer/5c949687a2e3a90445b8431f

Using U-net and public DAGM dataset (with Nvidia GPU T4, TRT5), it shows 23.5x perf. boost using T4/TRT5, compared to CPU-TF.

AGENDA

Industrial Defect Inspection

Nvidia GPU Cloud (NGC) Docker images

DL Model set up - Unet

Data preparation

Defect segmentation – precision/recall

Automatic Mixed Precision - AMP

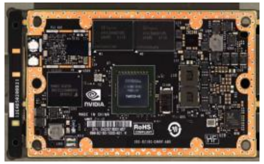
GPU accelerated inferencing – TF-TRT & TRT

INDUSTRIAL
DEFECT INSPECTION

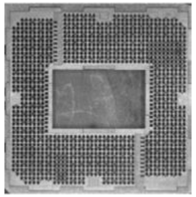
Industrial Inspection Use-case

Display panel

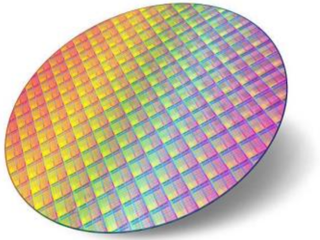
PCB



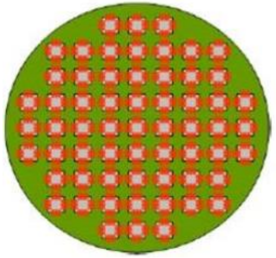
CPU socket



Foundry/Wafer



IC Packaging



Automotive
Manufacturing

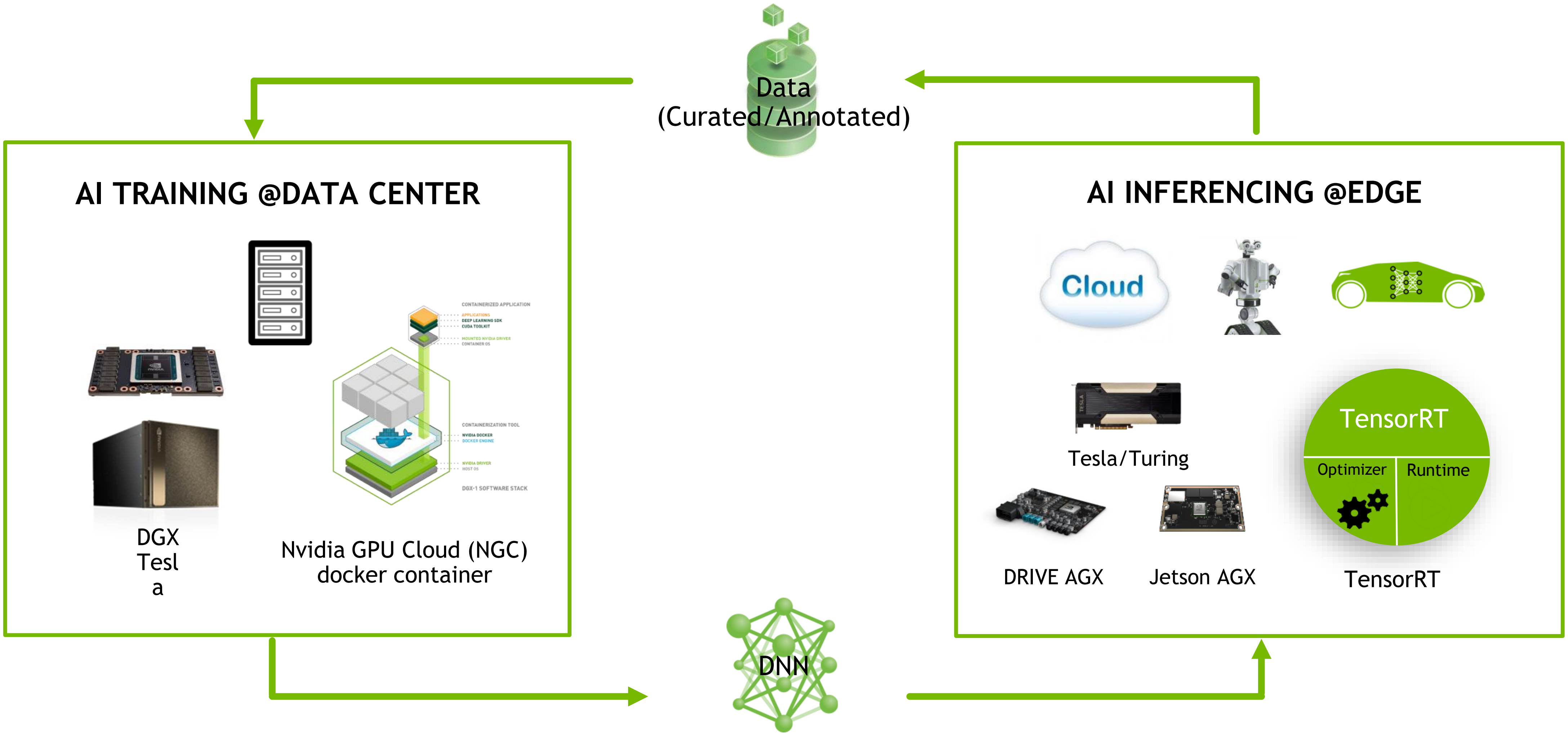
Battery
surface defects
(Electric car,
Mobile phone)

2 Main Scenarios
- Industrial/Manufacturing inspection

With AOI

Without AOI

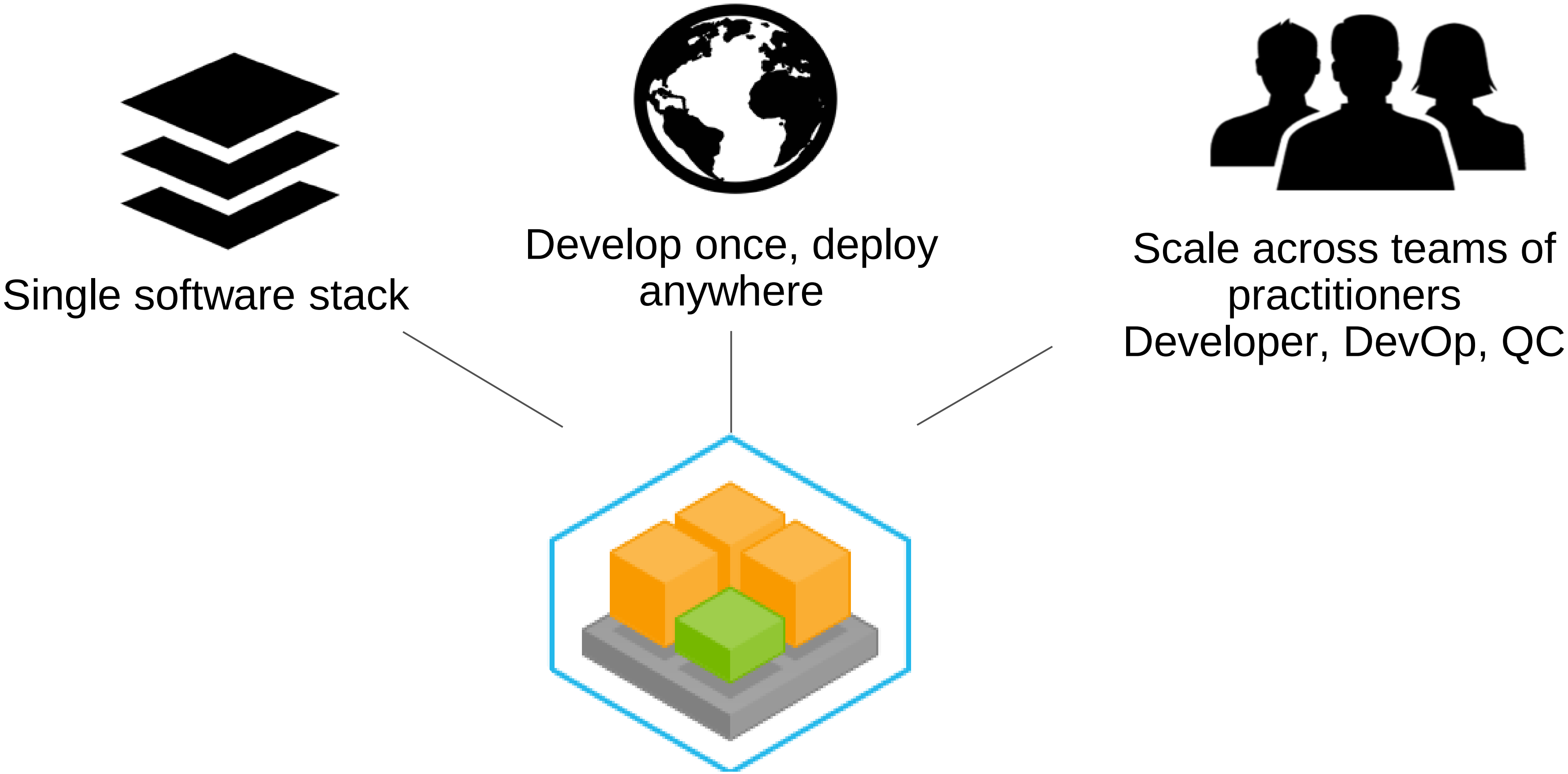
NVIDIA DEEP LEARNING PLATFORM



NGC DOCKER IMAGES

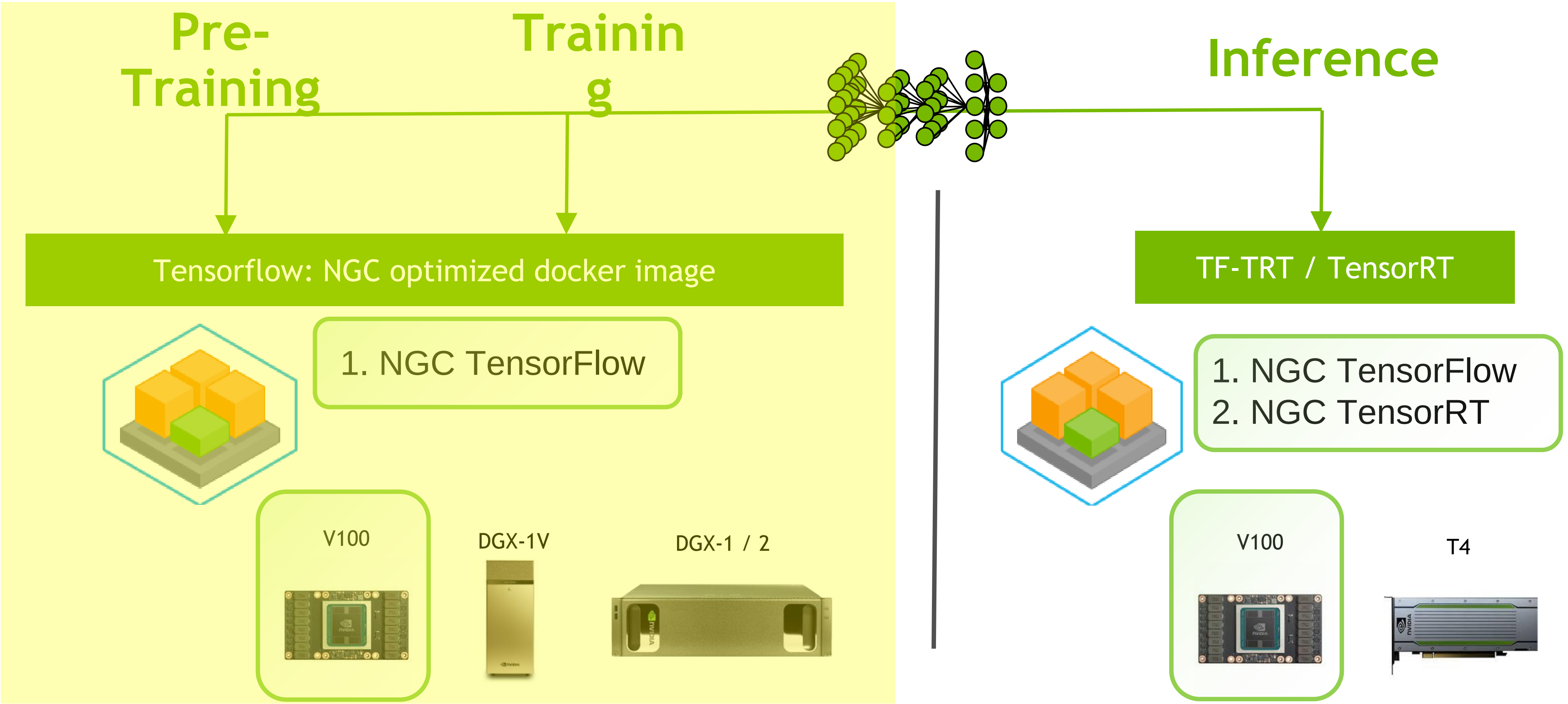
Benefits for Deep Learning Workflow

High Level Benefits and Feature Set



Defect classification workflow

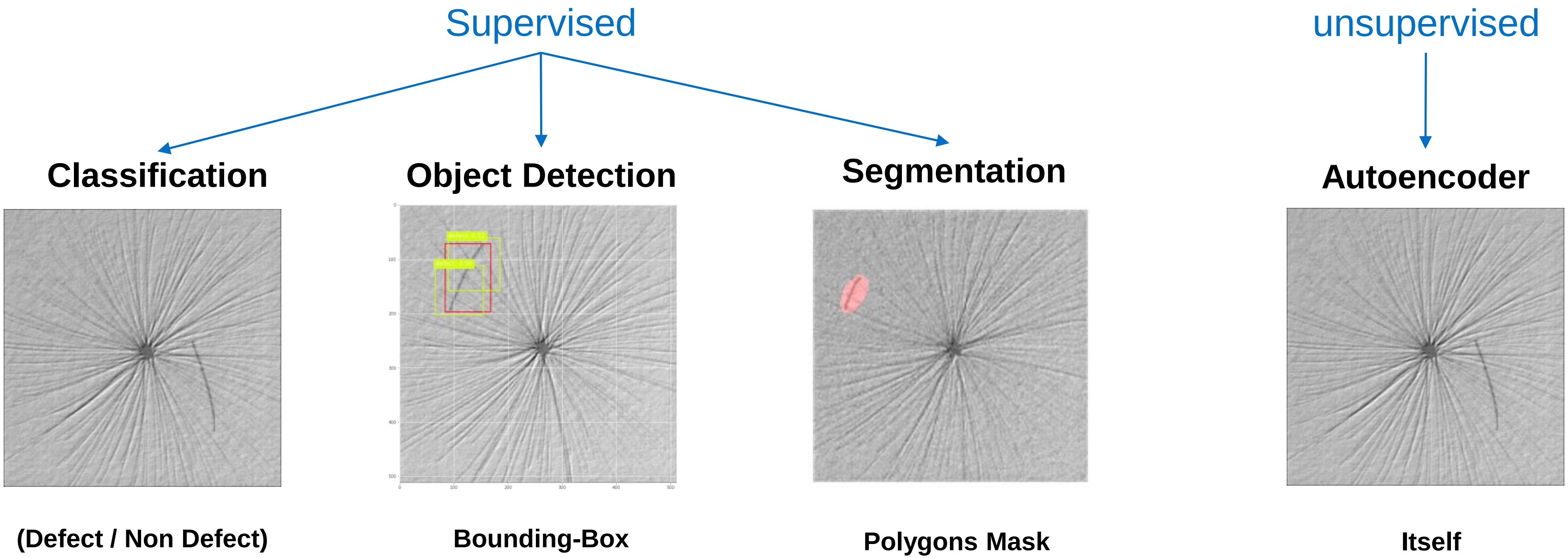
Rapid prototyping for production with NGC



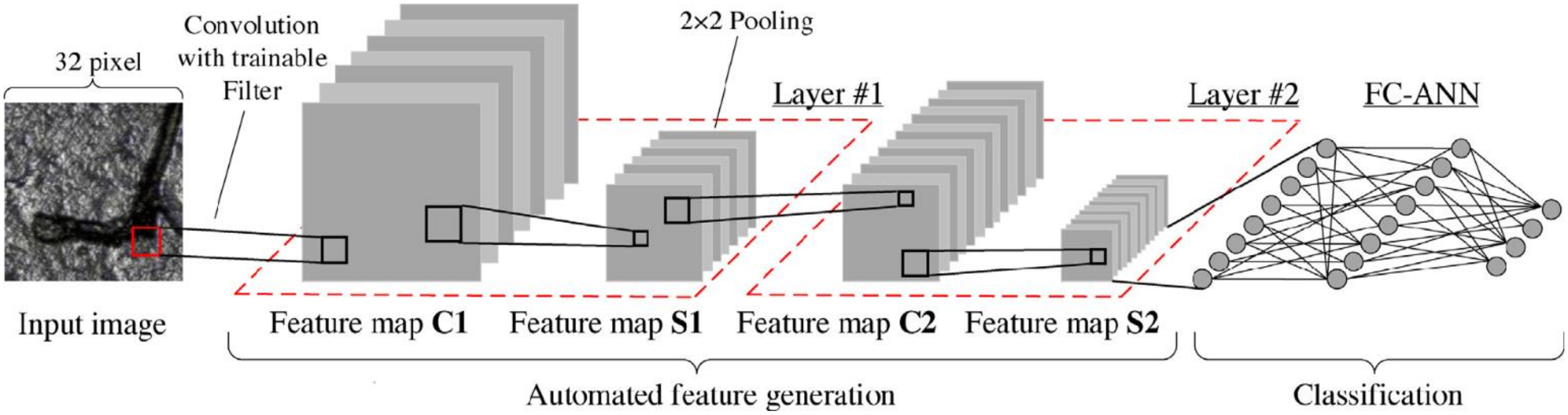
Used in industrial inspection white paper

MODEL SET UP

DL FOR DEFECT INSPECTION



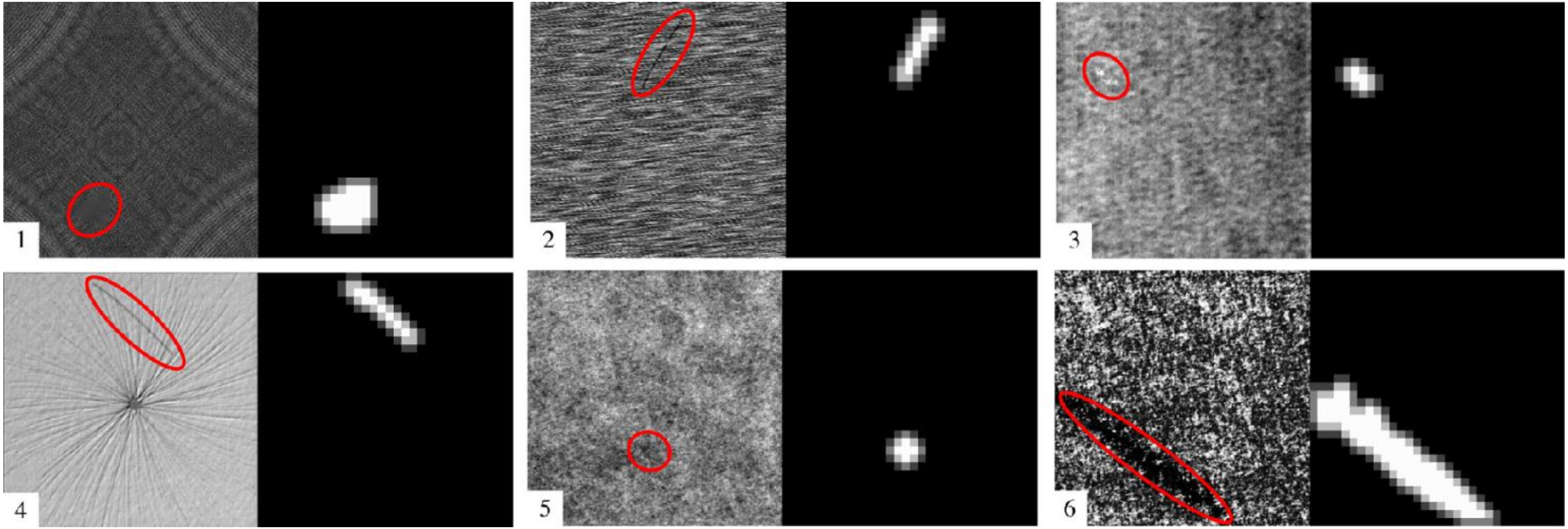
FROM LITERATURE: CNN/LENET (2016)



Source: Design of Deep Convolutional Neural Network Architectures for Automated Feature Extraction in Industrial Inspection, D. Weimer et al, 2016

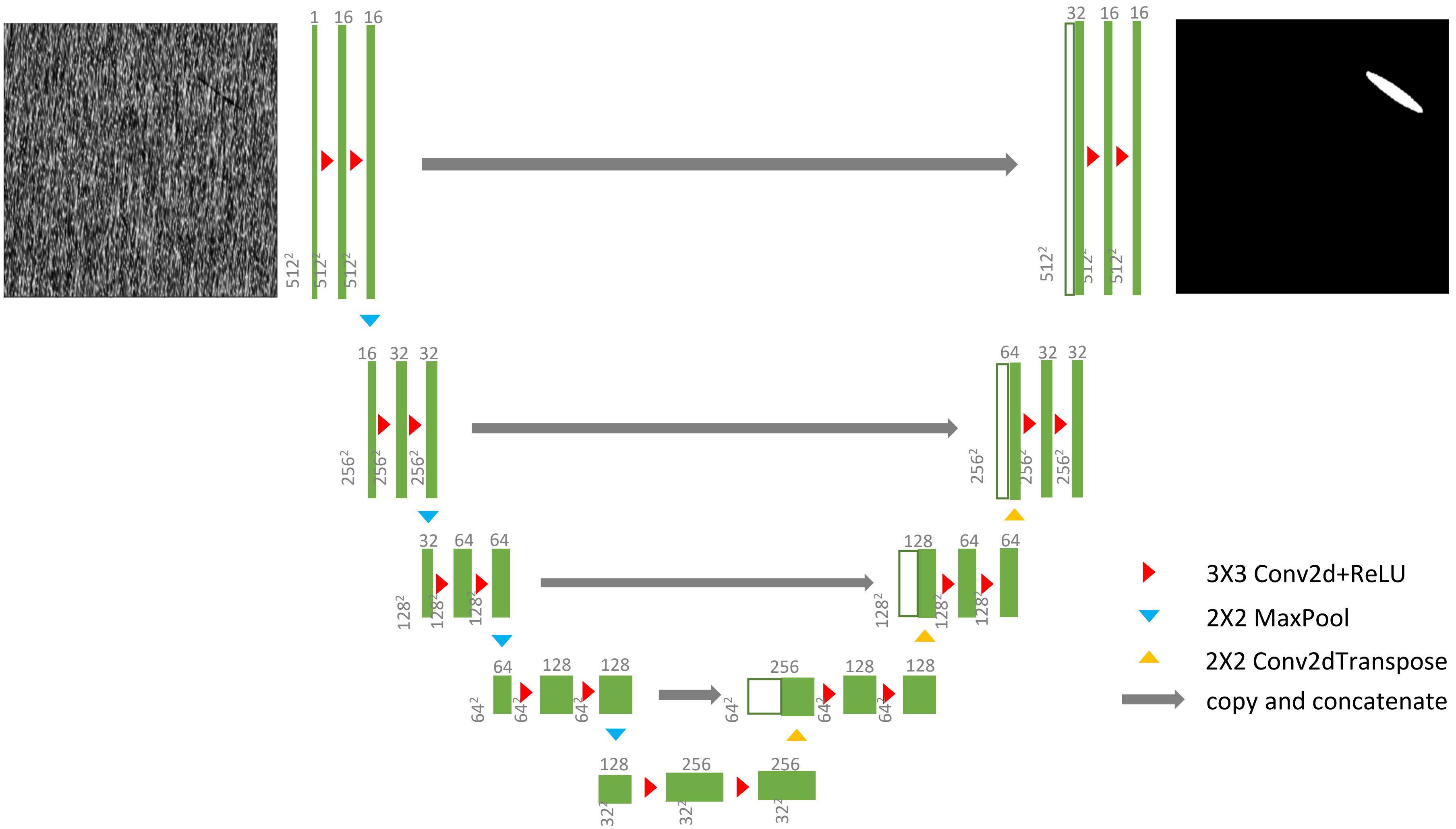
FROM LITERATURE CNN/LENET (2016)

Coarse segmentation results - can we do better?



Source: Design of Deep Convolutional Neural Network Architectures for Automated Feature Extraction in Industrial Inspection, D. Weimer et al, 2016

U-Net structure



KERAS-TF IMPLEMENTATION- ENCODING

Convolution

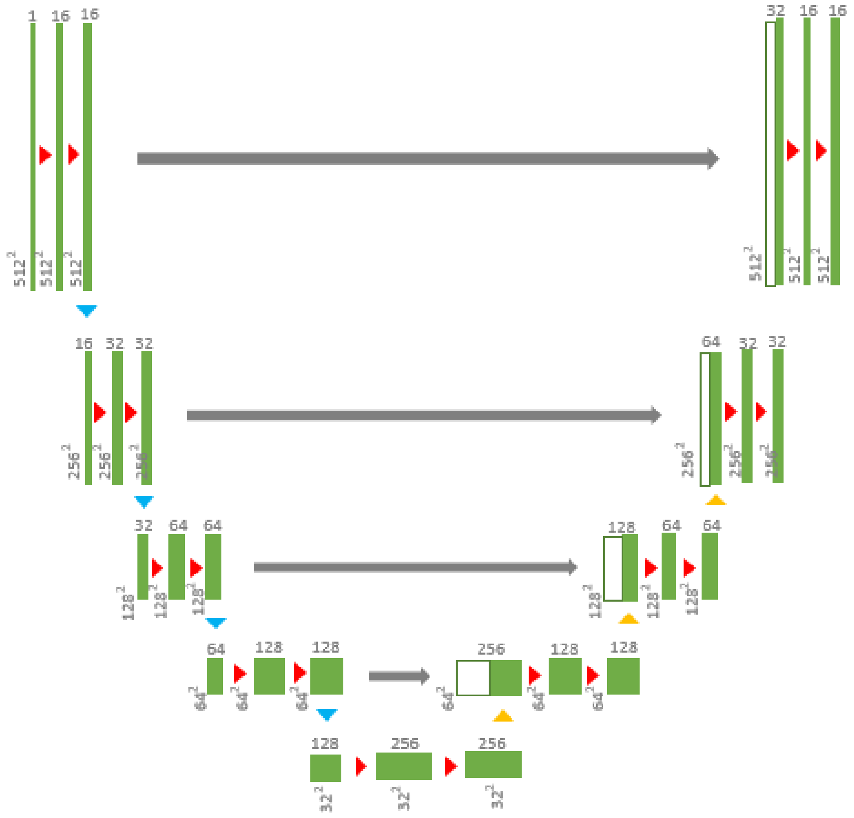
```
inputs = Input((IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS))
inputs_norm = Lambda(lambda x: x/127.5 - 1.)
conv1 = Conv2D(8, (3, 3), activation='relu', padding='same')(inputs)
conv1 = Conv2D(8, (3, 3), activation='relu', padding='same')(conv1)
pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

conv2 = Conv2D(16, (3, 3), activation='relu', padding='same')(pool1)
conv2 = Conv2D(16, (3, 3), activation='relu', padding='same')(conv2)
pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

conv3 = Conv2D(32, (3, 3), activation='relu', padding='same')(pool2)
conv3 = Conv2D(32, (3, 3), activation='relu', padding='same')(conv3)
pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

conv4 = Conv2D(64, (3, 3), activation='relu', padding='same')(pool3)
conv4 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv4)
pool4 = MaxPooling2D(pool_size=(2, 2))(conv4)

conv5 = Conv2D(128, (3, 3), activation='relu', padding='same')(pool4)
conv5 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv5)
```



KERAS-TF IMPLEMENTATION- ENCODING

deconvolution

```
up6 = merge([UpSampling2D(size=(2, 2))(conv5), conv4], mode='concat', concat_axis=3)
conv6 = Conv2D(64, (3, 3), activation='relu', padding='same')(up6)
conv6 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv6)

up7 = merge([UpSampling2D(size=(2, 2))(conv6), conv3], mode='concat', concat_axis=3)
conv7 = Conv2D(32, (3, 3), activation='relu', padding='same')(up7)
conv7 = Conv2D(32, (3, 3), activation='relu', padding='same')(conv7)

up8 = merge([UpSampling2D(size=(2, 2))(conv7), conv2], mode='concat', concat_axis=3)
conv8 = Conv2D(16, (3, 3), activation='relu', padding='same')(up8)
conv8 = Conv2D(16, (3, 3), activation='relu', padding='same')(conv8)

up9 = merge([UpSampling2D(size=(2, 2))(conv8), conv1], mode='concat', concat_axis=3)
conv9 = Conv2D(8, (3, 3), activation='relu', padding='same')(up9)
conv9 = Conv2D(8, (3, 3), activation='relu', padding='same')(conv9)

conv10 = Conv2D(1, (1, 1), activation='sigmoid')(conv9)

model = Model(inputs=inputs, outputs=conv10)
```

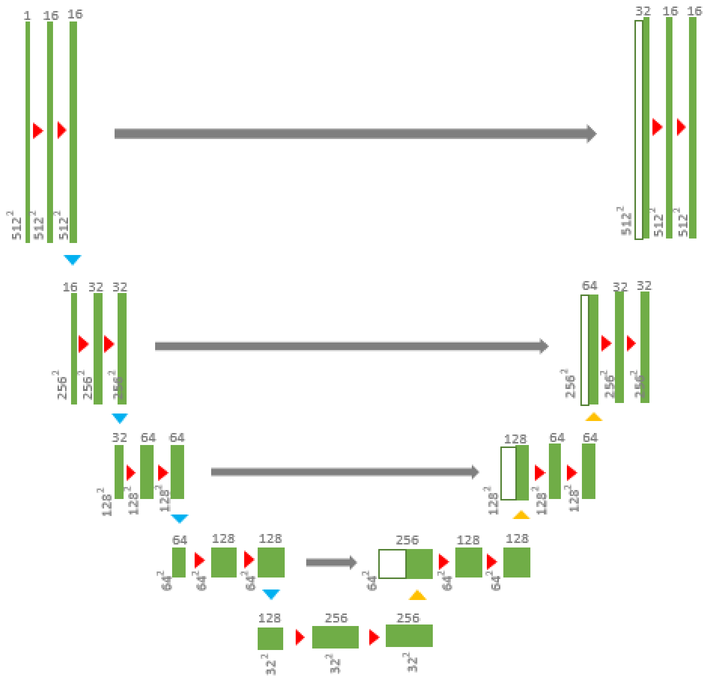
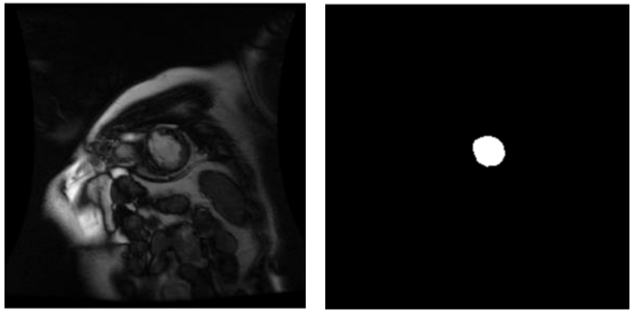


Image segmentation on medical images

Same process among various use cases

Data Science BOWL
2016

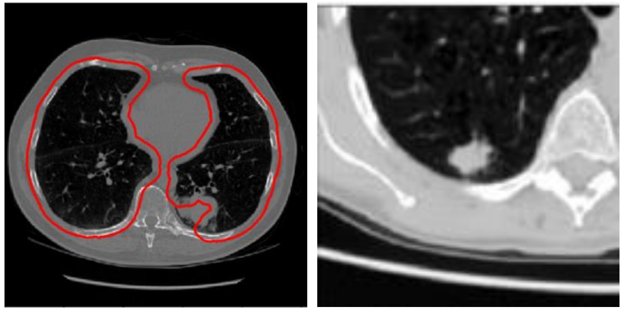


MRI image

Left ventricle

heart disease

Data Science BOWL
2017

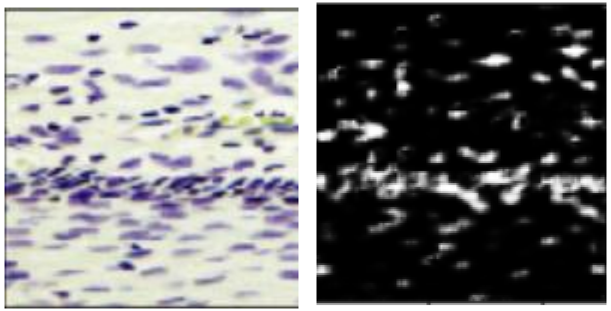


CT image

Nodule

Lung cancer

Data Science BOWL
2018



Image

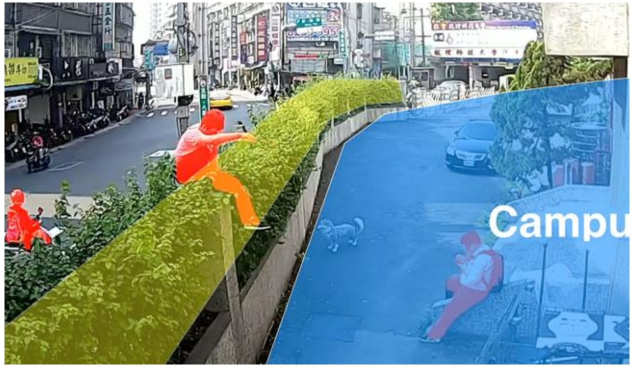
Nuclei

Drug discovery

Many others

Different verticals

Surveillance



Human

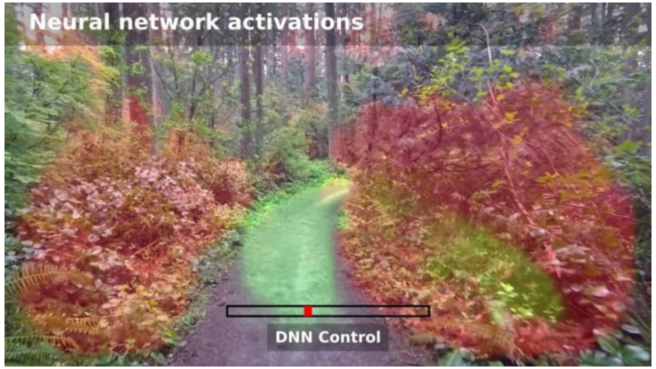
Anomaly Detection

Autonomous Car

Road Space

Space for Self Driving Car

Drone

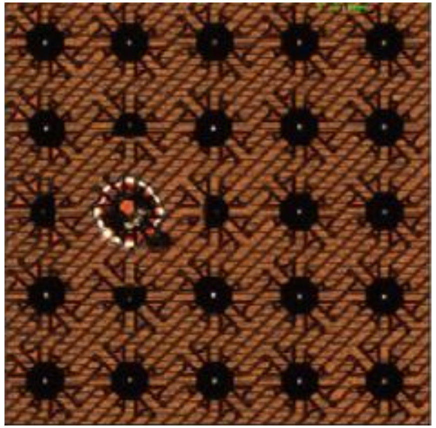


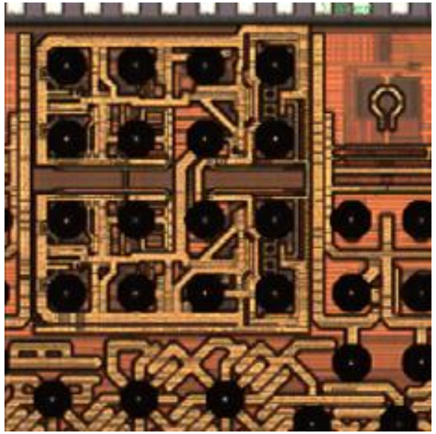
Path Space

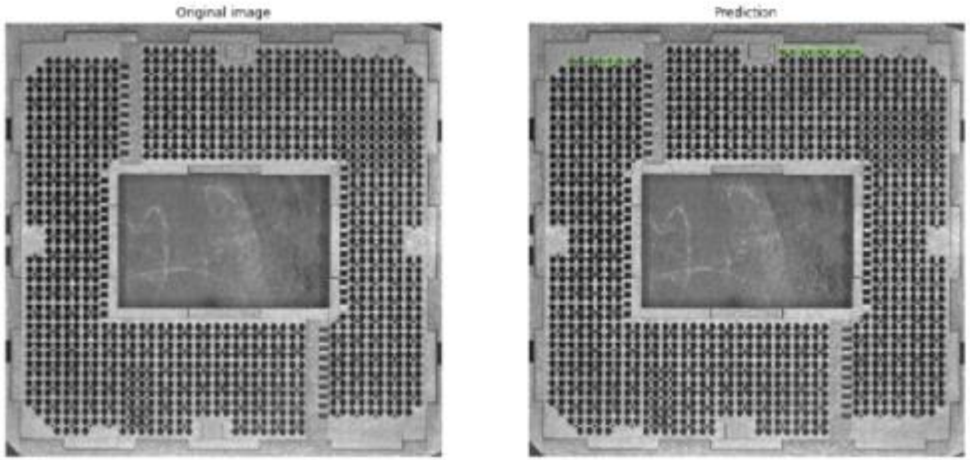
Navigation

MANUFACTURING

Defect Inspection



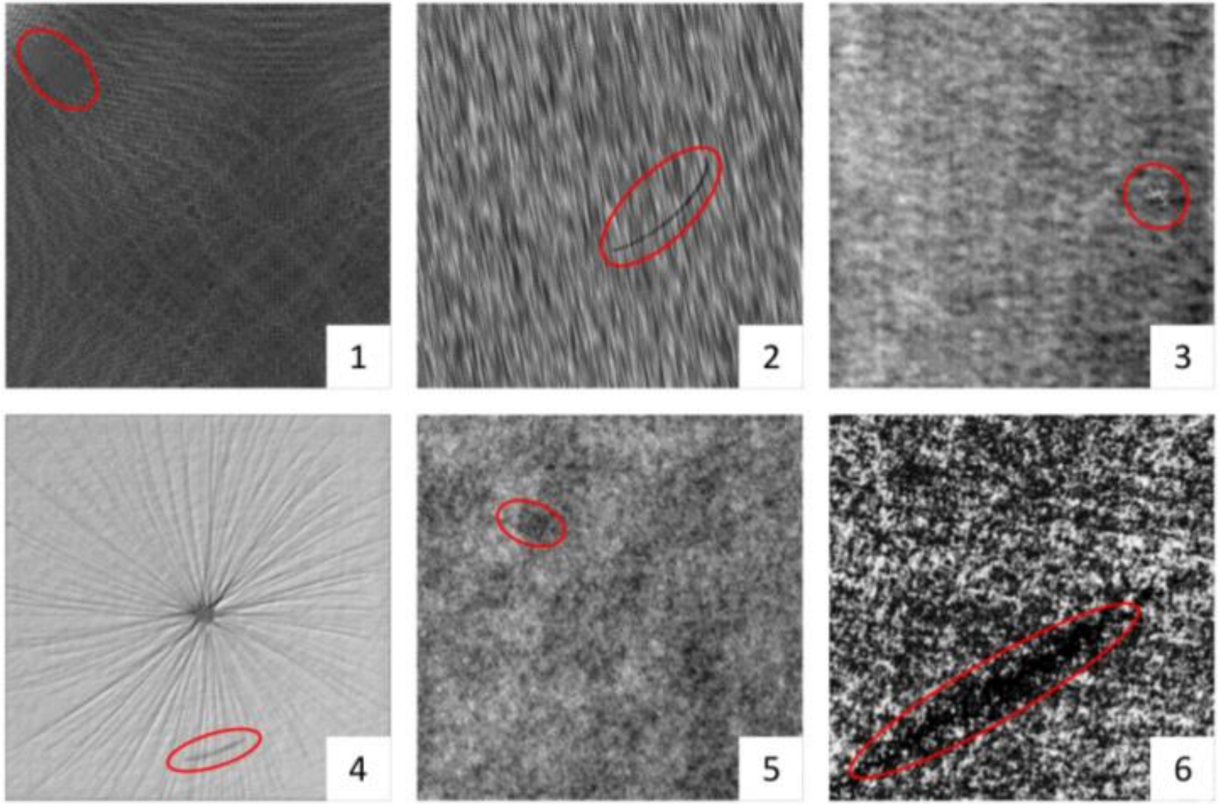




DATA PREPARATION

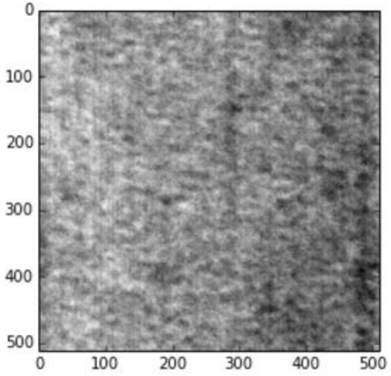
DATASET FOR INDUSTRIAL OPTICAL INSPECTION

DAGM (from German Association for Pattern Recognition)

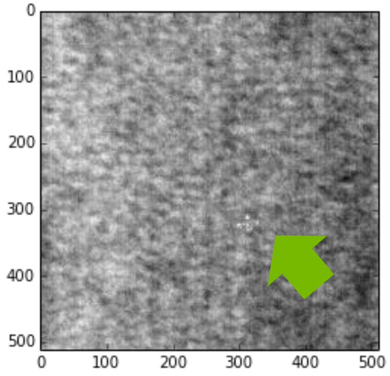


- http://resources.mpi-inf.mpg.de/conferences/dagm/2007/prizes.html

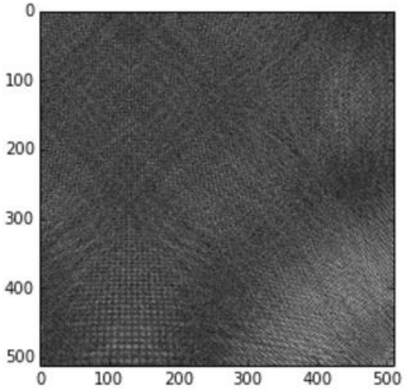
DAGM DATASET



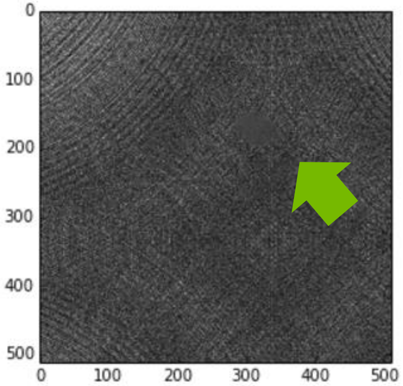
Pass



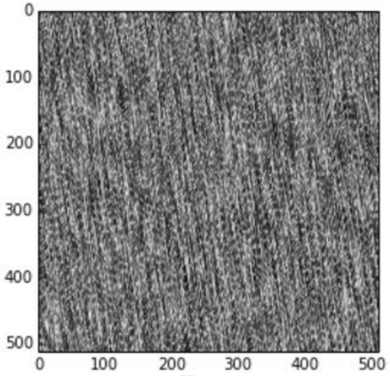
NG



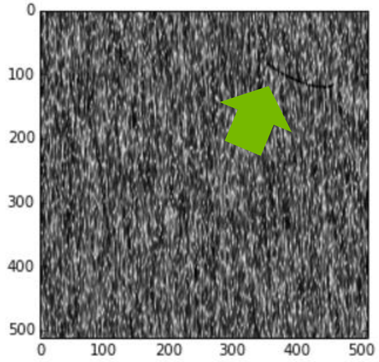
Pass



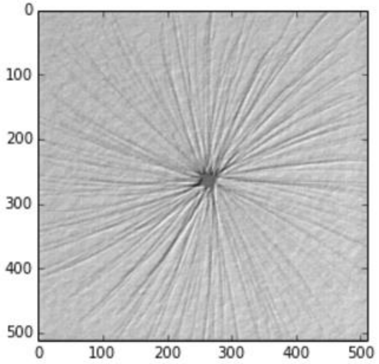
NG



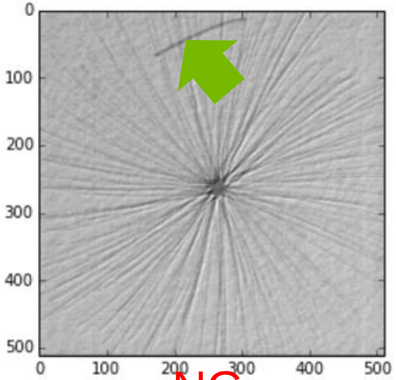
Pass



NG



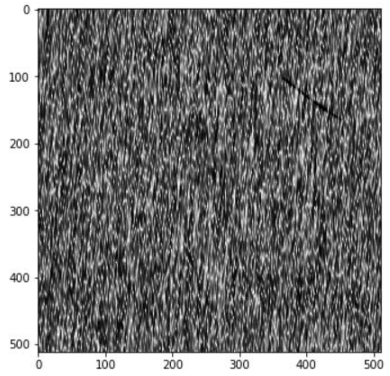
Pass

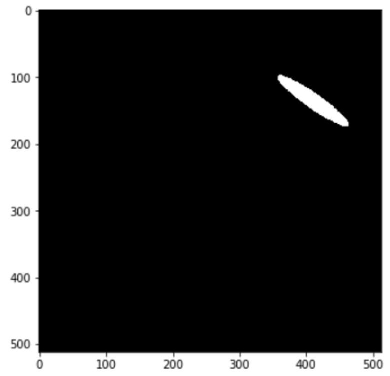


NG

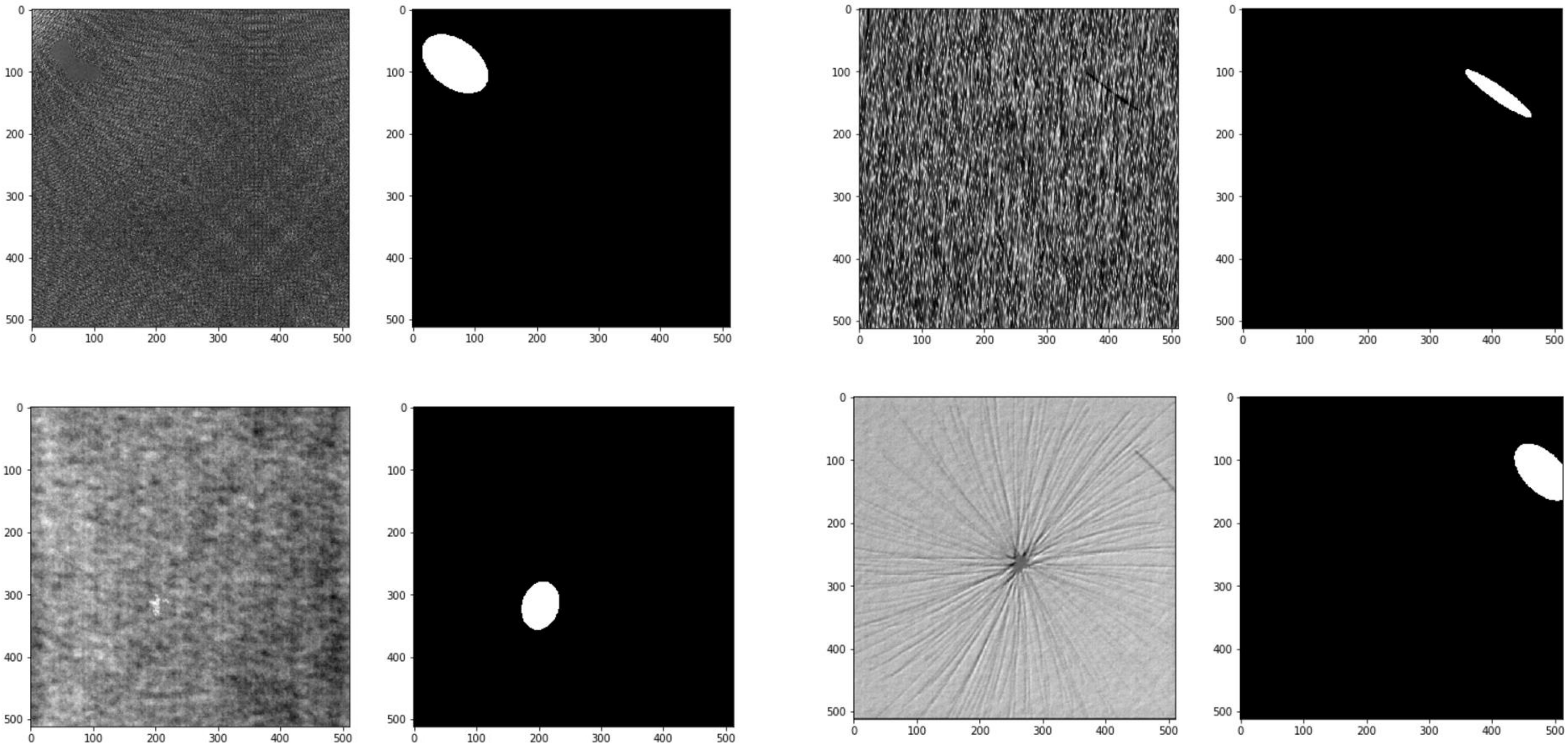
DAGM DETAILS

- Original images are 512 x 512 grayscale format
- Output is a tensor of size 512 x 512 x 1
  - Each pixel belongs to one of two classes
  - 6 defect classes
- Training set consist of 100 defect images
- Validation set consist of 50 defect images





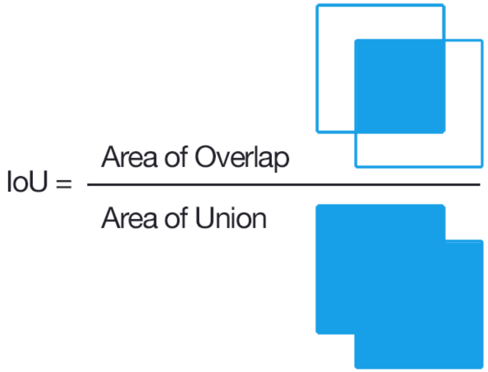
DAGM EXAMPLES WITH LABELS



Dice Metric (IOU) for unbalanced dataset

- Metric to compare the similarity of two samples:

$$\frac{2A_{nl}}{A_n + A_l}$$

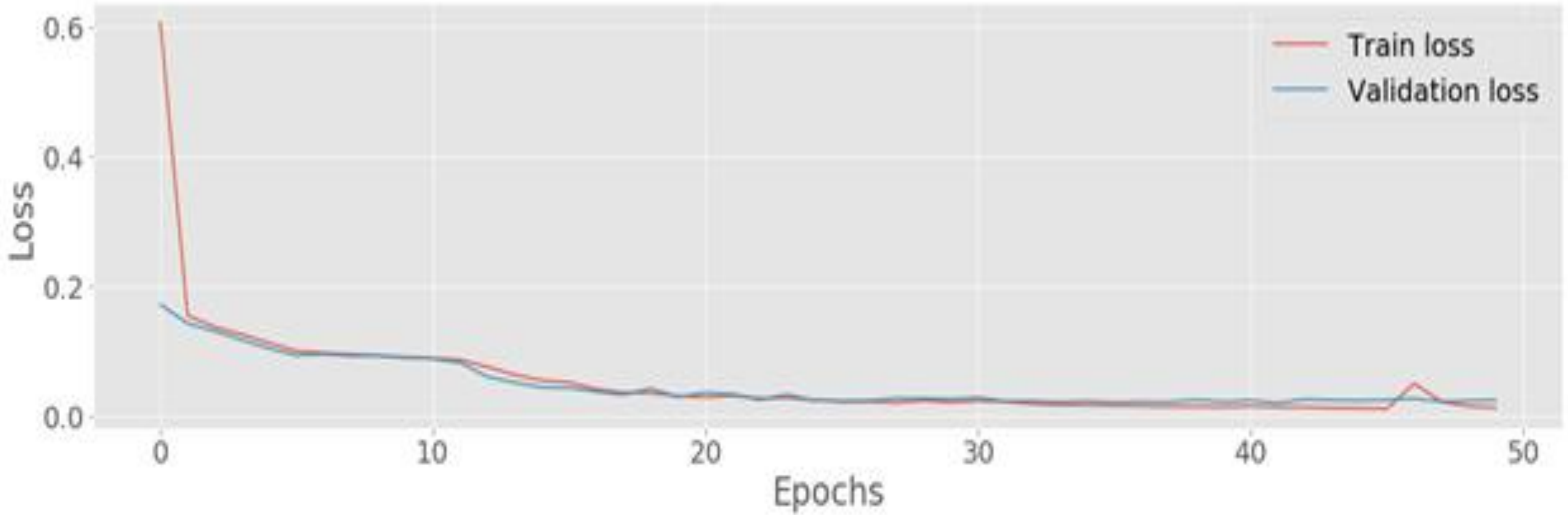


  - Where:
    - A_n is the area of the contour predicted by the network
    - A_l is the area of the contour from the label
    - A_{nl} is the intersection of the two
      - The area of the contour that is predicted correctly by the network
      - 1.0 means perfect score.

- More accurately compute how well we're predicting the contour against the label

- We can just count pixels to give us the respective areas

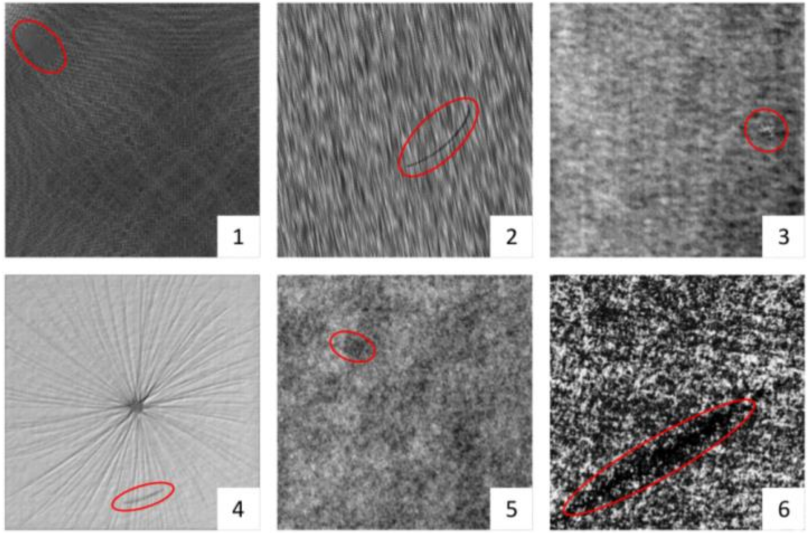
LEARNING CURVES

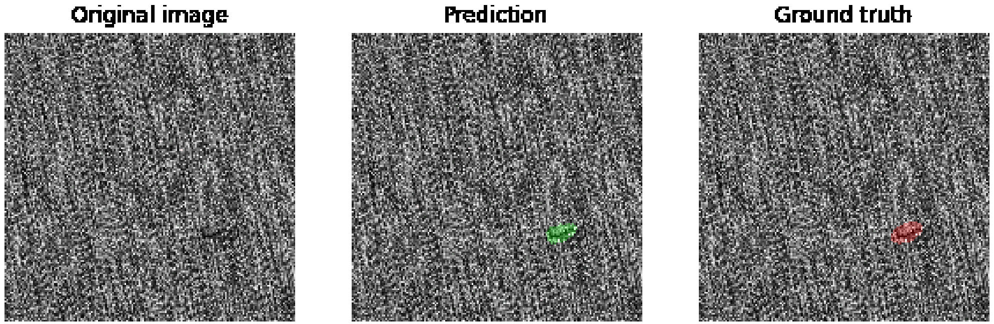


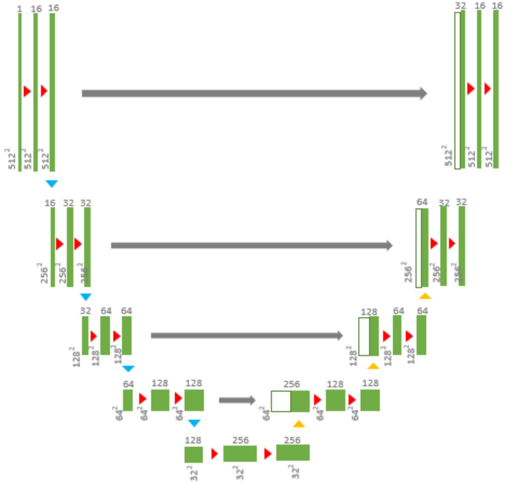
U-NET / DAGM FOR INDUSTRIAL INSPECTION

- **DAGM merged binary classification dataset: 6000** defect-free, **132** defect images

- **Challenges**: Not all deviations from the texture are necessarily defects.

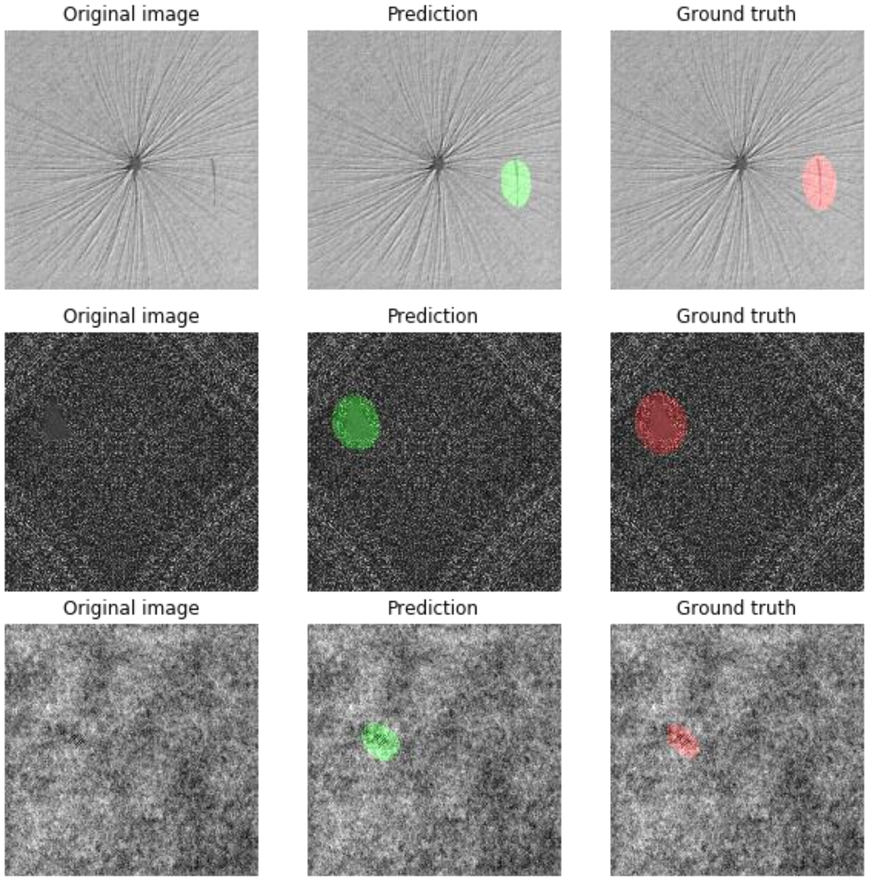






DEFECT SEGMENTATION - PRECISION/RECALL

FINAL DECISION



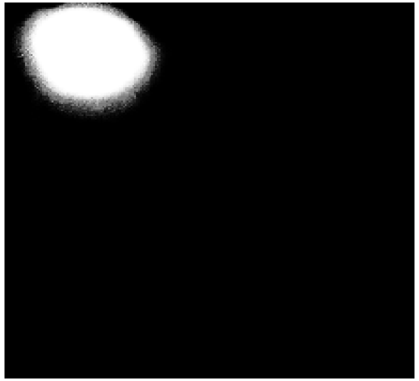
DEFECT VS NON-DEFECT BY THRESHOLDING

Segmentation model outputs Numpy array of class probability of each class (example 2 classes)

Declare as defect (white) if probability is higher than threshold (=0.5)

```
array([[ 6.17885776e-03,   3.82234044e-02,   9.50025606e-06, ...,
         4.50918742e-05,   3.49759248e-05,   3.65408661e-04],
       [ 6.45390755e-05,   4.58258086e-07,   2.12041887e-05, ...,
         3.15845439e-09,   1.69029056e-06,   1.10975248e-04],
       [ 2.03725667e-05,   4.74613626e-06,   6.89793808e-07, ...,
         6.43013749e-08,   1.97115969e-06,   2.85665534e-04],
       ...,
       [ 2.50566706e-10,   4.80150497e-08,   2.86757146e-10, ...,
         9.31098111e-06,   2.05957076e-05,   4.73519601e-03],
       [ 1.80557666e-10,   1.41850676e-09,   1.18475485e-09, ...,
         2.04379503e-05,   3.10234725e-03,   4.20572087e-02],
       [ 1.74140851e-07,   1.64427387e-08,   2.98866799e-11, ...,
         3.39166650e-06,   1.28269540e-02,   2.99611967e-02]], dtype=float32)
```

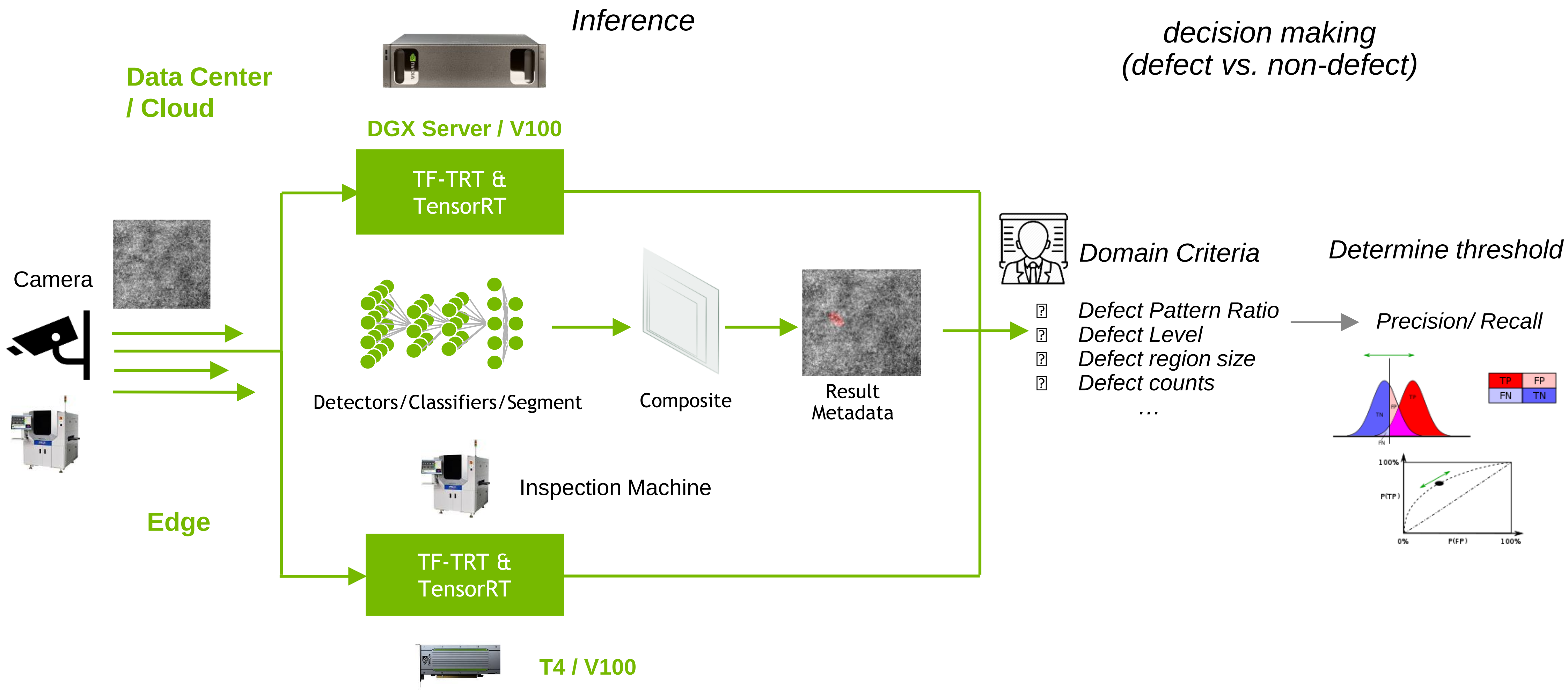
Thresholding



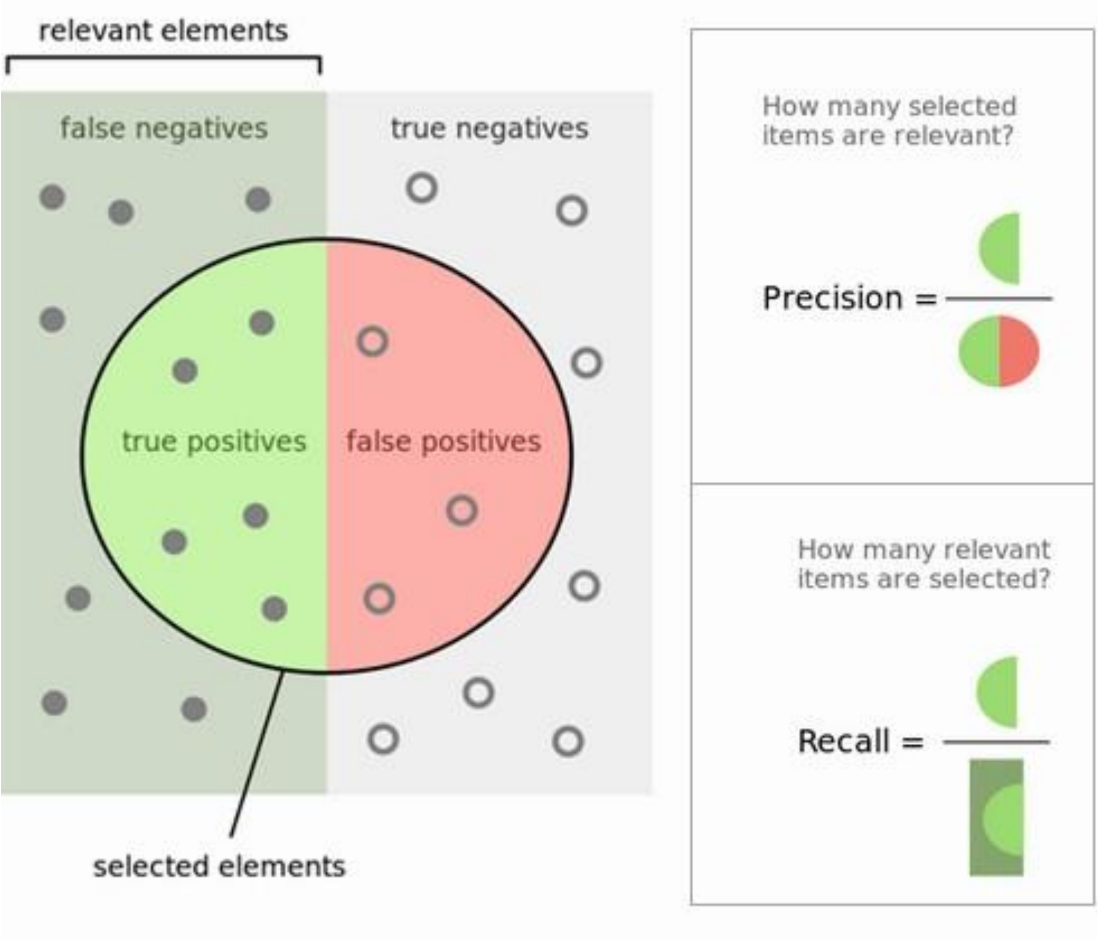
query image 512x512

INFERENCE PIPELINE

Domain expertise involved decision making (not a black-box)



(Example) Precision/Recall diagram

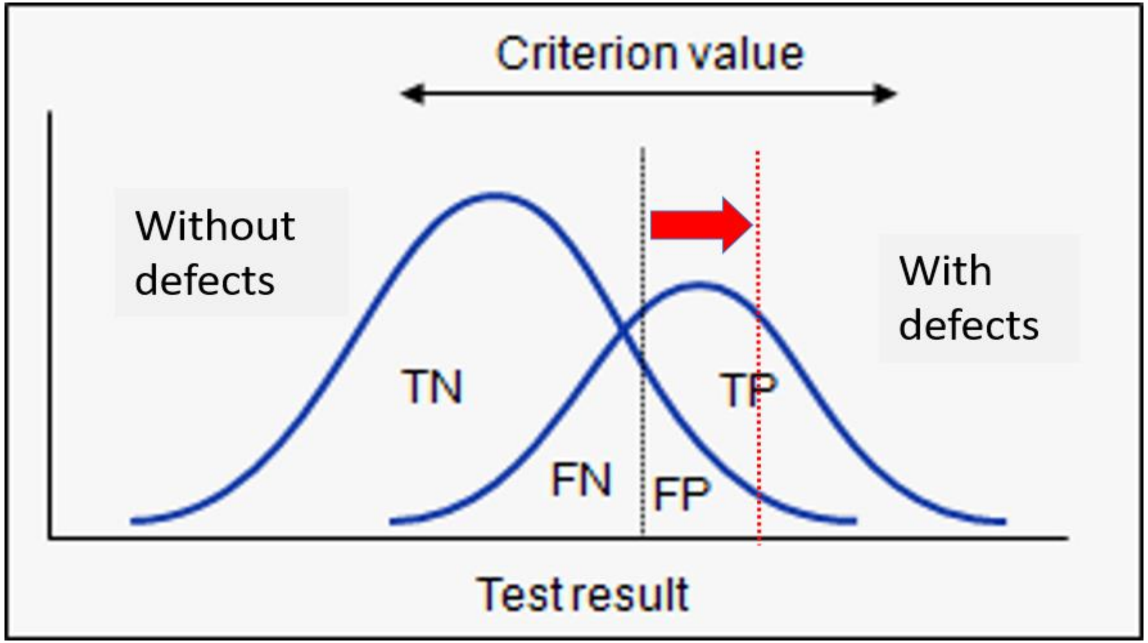


(Example) Simple binary anomaly detector

Threshold of probability of defect: higher number means harder for classifier to detect as defect class.

Higher threshold: FP lower, precision (TP/(TP+FP)) higher

FN higher, recall (TP/(TP+FN)) lower



TP: True Positive, FP: False Positive, FN: False Negative, TN: True Negative.

red arrow means moving threshold of probability on defect detection into higher value.

Precision/Recall Results

Experimental results verifies precision/recall trade-off.

Domain expert knowledge involved: choose threshold per your application and business needs

threshold	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
TP	137	135	135	135	135	135	135	133	131
TN	885	893	899	899	899	899	899	900	901
FP	16	8	2	2	2	2	2	1	0
FN	1	3	3	3	3	3	3	5	7
FP rate	0.0178	0.0089	0.0023	0.0023	0.0023	0.0023	0.0023	0.0011	0.0000
precision	0.8954	0.9441	0.9854	0.9854	0.9854	0.9854	0.9854	0.9925	1.0000
recall	0.9928	0.9783	0.9783	0.9783	0.9783	0.9783	0.9783	0.9638	0.9493

Choose: threshold = 0.8 for high precision = 0.9925 & small FP rates = 0.0011

Precision/Recall - reducing false positives

Precision =TP/(TP+FP) : 99.25%

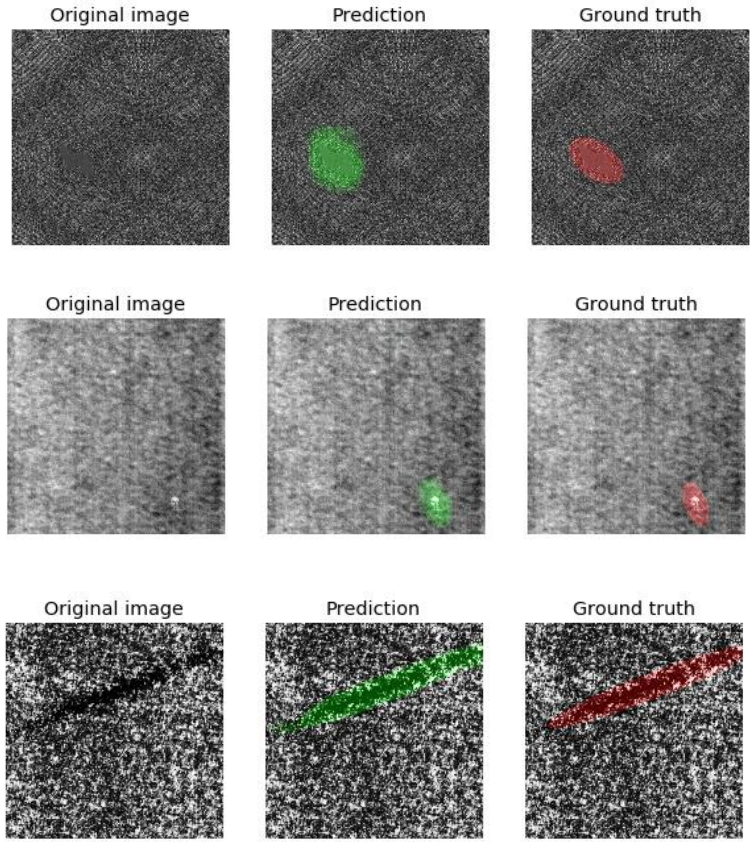
Recall = TP/(TP+FN) : 96.38%

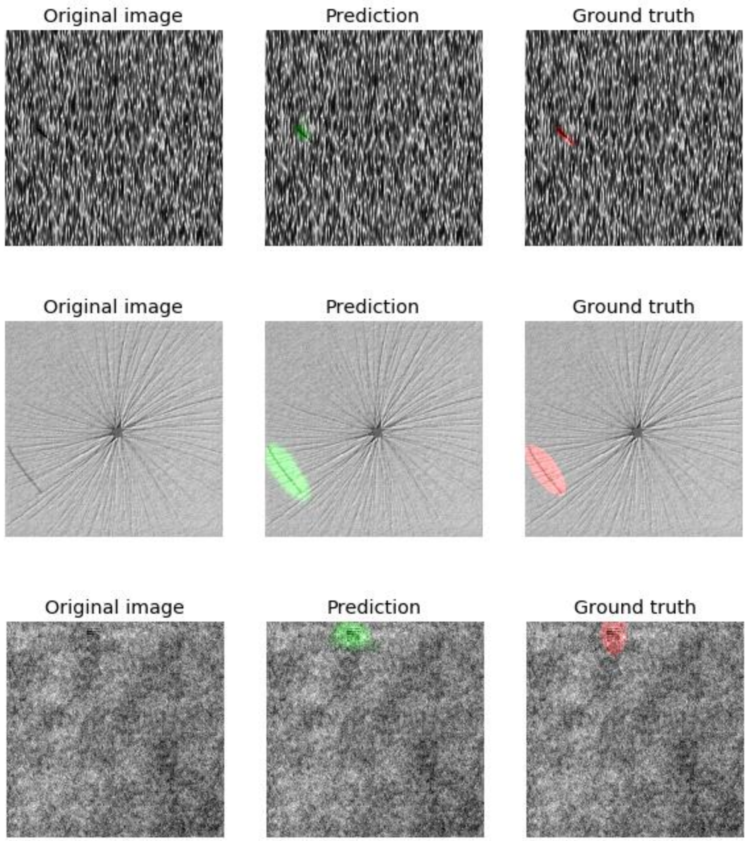
False alarm rate = FP/(FP+TN): 0.11%

		Actual	
		defect	defect free
Predict	defect	99.25% (TP)	0.75% (FP)
	defect free	0.55% (FN)	99.45% (TN)

*sensitivity=recall=true positive rate,
specificity=true negative rate=TN/(TN+FP), false alarm rate=false positive rate

Defect segmentation (U-net + Thresholding)





AUTOMATIC MIXED PRECISION FOR U-NET ON V100

TENSOR CORES FOR DEEP LEARNING

Mixed Precision implementation using Tensor Cores on Volta and Turing GPUs

Tensor Cores

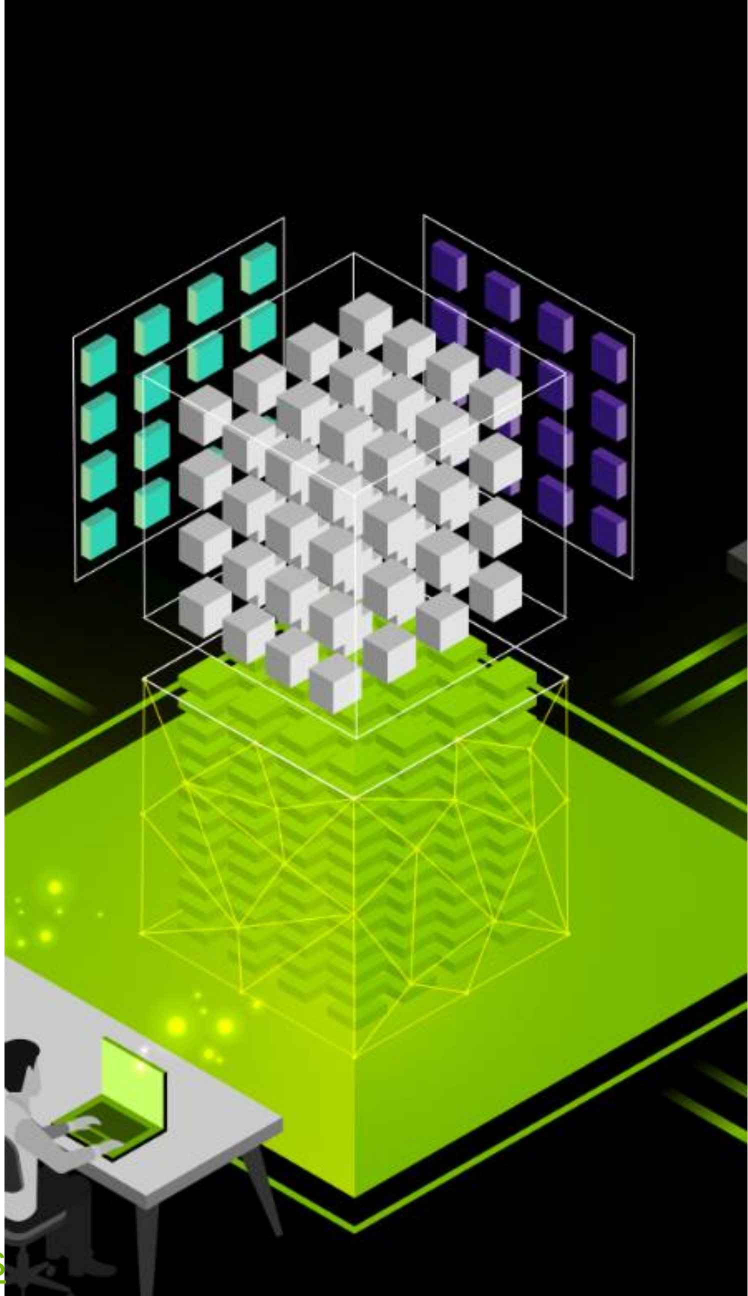
- A revolutionary technology that accelerates AI performance by enabling efficient mixed-precision implementation
- Accelerate large matrix multiply and accumulate operations in a single operation

Mixed Precision Technique
combined use of different numerical precisions in a computational method; focus is on FP16 and FP32 combination.

Benefits

- Decreases the required amount of memory enabling training of larger models or training with larger mini-batches
- Shortens the training or inference time by lowering the required resources by using lower-precision arithmetic

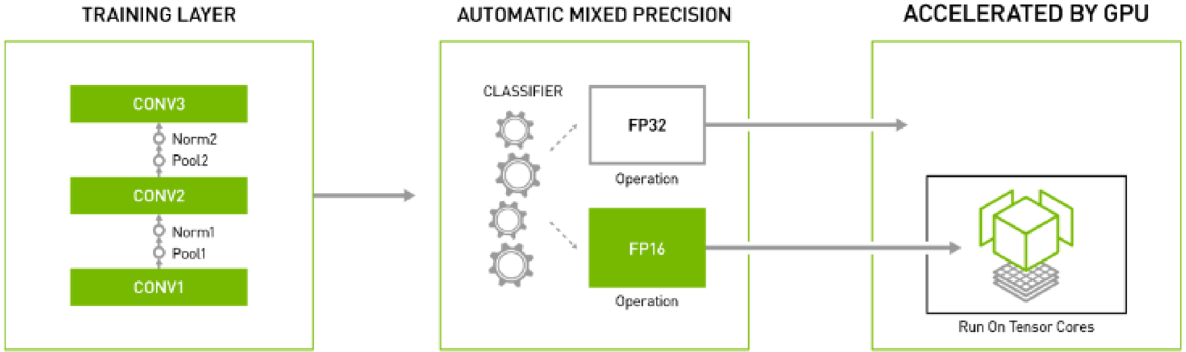
https://developer.nvidia.com/tensor-cores



Automatic Mixed Precision

Easy to Use, Greater Performance and Boost in Productivity

- Insert two lines of code to introduce Automatic Mixed-Precision in your training layers for up to a 3x performance improvement.
- The Automatic Mixed Precision feature uses a graph optimization technique to determine FP16 operations and FP32 operations.
- Available in TensorFlow, PyTorch and MXNet via our NGC Deep Learning Framework Containers.



More details: https://developer.nvidia.com/automatic-mixed-precision

Unleash the next generation AI performance and get faster to the market!

Enable Automatic Mixed Precision

Add Just A Few Lines of Code, Get Upto 3X Speedup

TensorFlow

```
os.environ['TF_ENABLE_AUTO_MIXED_PRECISION'] = '1'
```

OR thru NGC

```
export TF_ENABLE_AUTO_MIXED_PRECISION=1
```

PyTorch

```
model, optimizer = amp.initialize(model, optimizer)

with amp.scale_loss(loss, optimizer) as
scaled_loss:

    scaled_loss.backward()
```

MXNet

```
amp.init()
amp.init_trainer(trainer)
with amp.scale_loss(loss, trainer) as scaled_loss:
   autograd.backward(scaled_loss)
```

More details: https://developer.nvidia.com/automatic-mixed-precision

U-Net AMP performance boost

Training performance (17% boost)

# GPUs	Precision	Training (Imgs/sec)	Training Time	Speedup
1	FP32	89	7m44	1.00
1	Automatic Mixed Precision (AMP)	104	6m40	1.17

Inference performance (30% boost)

# GPUs	Precision	Training (Imgs/sec)	Speedup
1	FP32	228	1.00
1	Automatic Mixed Precision (AMP)	301	1.32

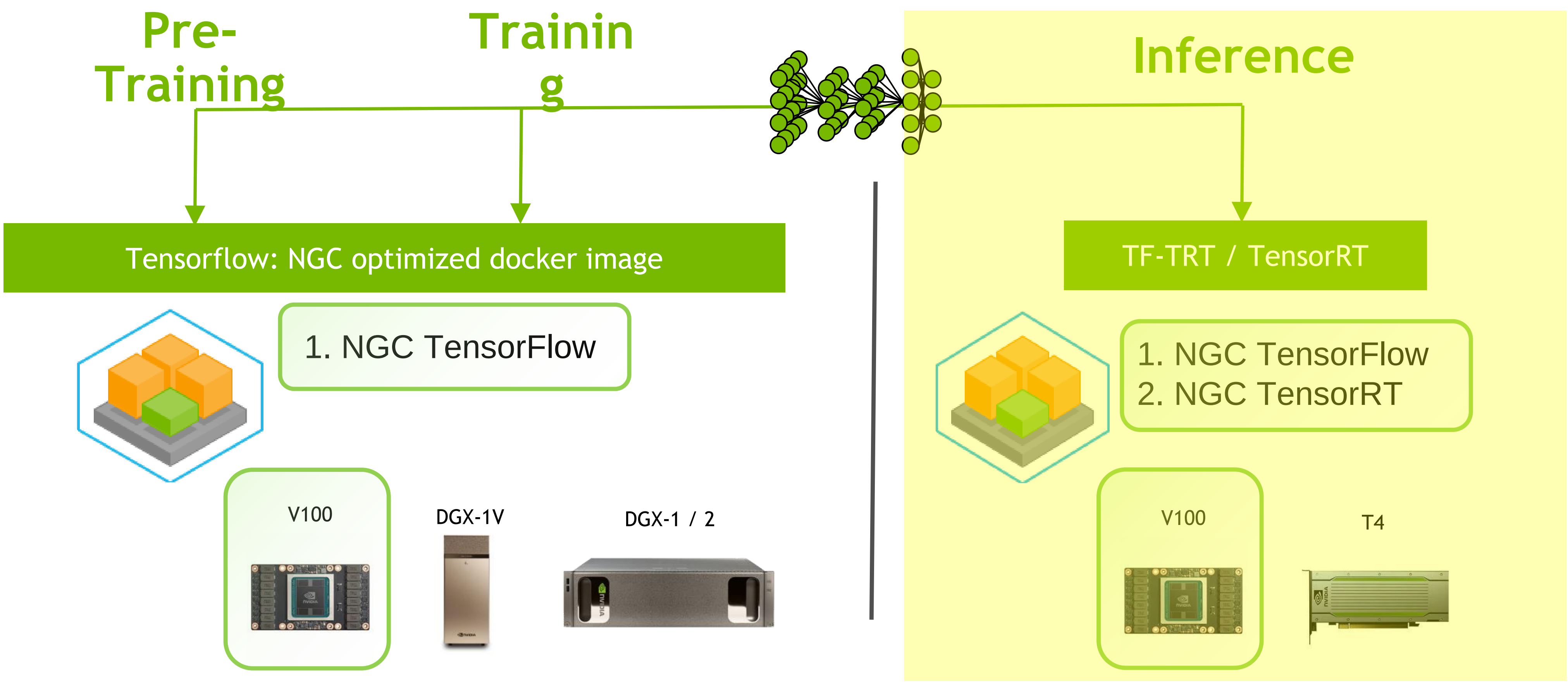
https://github.com/NVIDIA/DeepLearningExamples/blob/master/TensorFlow/Segmentation/UNet_Industrial/README.md#training-accuracy-results

Courtesy of Jonathan Dekhtiar, Alex Fit-Flora at Nvidia

GPU-ACCELERATED INFERENCING

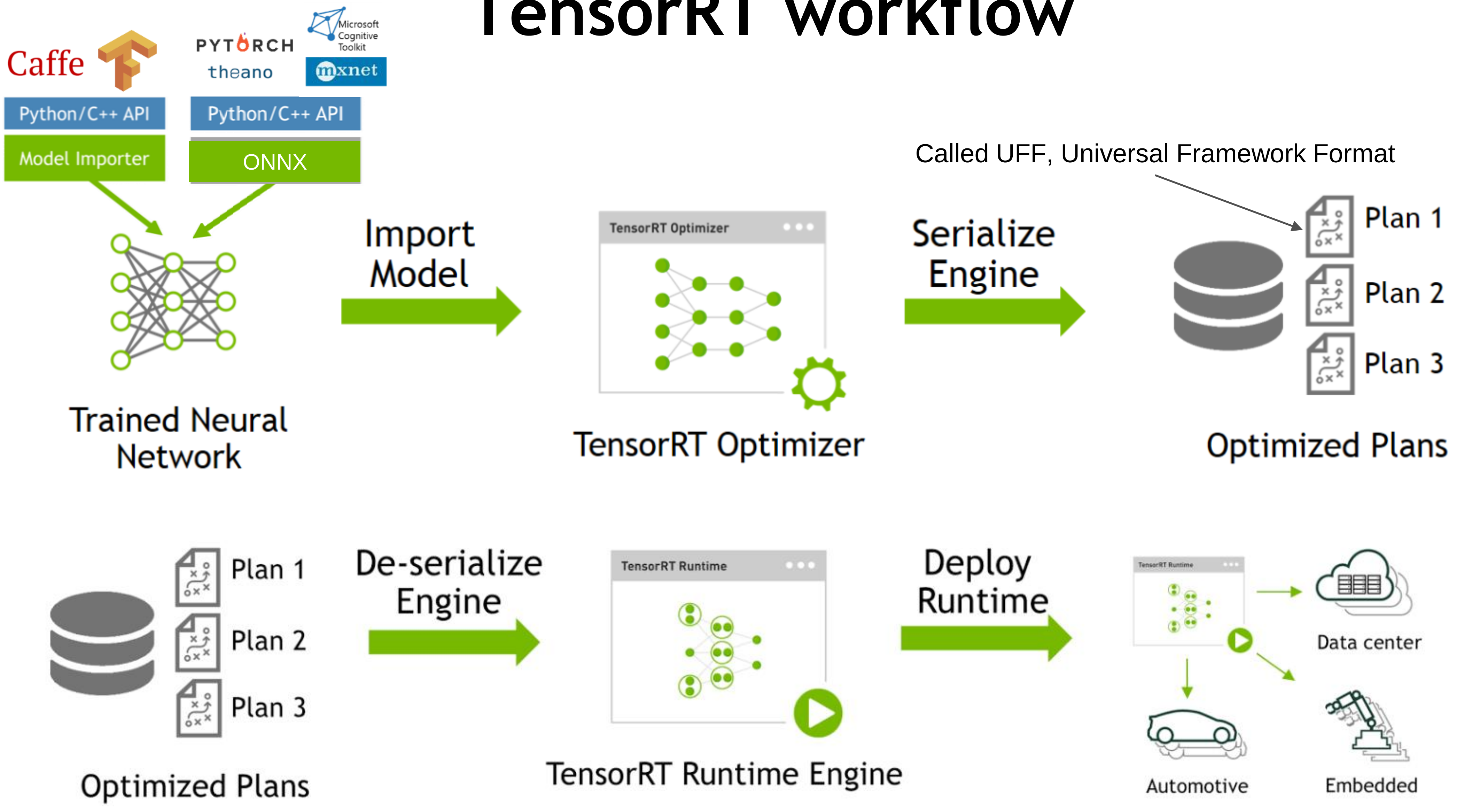
Defect classification workflow

Rapid prototyping for production with NGC



Used in industrial inspection white paper

TensorRT workflow



TensorRT Integrated With TensorFlow

Speed Up TensorFlow Inference With TensorRT Optimizations

Speed up TensorFlow model inference with TensorRT with new TensorFlow APIs

Simple API to use TensorRT within TensorFlow easily

Sub-graph optimization with fallback offers flexibility of TensorFlow and optimizations of TensorRT

Optimizations for FP32, FP16 and INT8 with use of Tensor Cores automatically

```
# Apply TensorRT optimizations
trt_graph = trt.create_inference_graph(frozen_graph_def,
                output_node_name,
                max_batch_size=batch_size,
                max_workspace_size_bytes=workspace_size,
                precision_mode=precision)

# INT8 specific graph conversion
trt_graph = trt.calib_graph_to_infer_graph(calibGraph)
```

Available from TensorFlow 1.7

https://github.com/tensorflow/tensorflow

V100/TRT4 Inference Results on U-net

TF-TRT for fast prototyping, TRT for maximum performance

8.6x speed-up by native TRT (FP16 precision)

Inference method		GPU-TF	TF-TRT	TRT
FP 32 bit	images/sec	141.8	236.1	1079.8
	perf. Increase	1	1.7	7.6
FP 16 bit*	images/sec	N/A	297.4	1219.7
	perf. Increase	1	2.1	8.6

FP 16 bit*: by mixed precision TensorCore in V100 GPU

TESLA T4

WORLD'S MOST ADVANCED SCALE-OUT GPU

320 Turing Tensor Cores
2,560 CUDA Cores
65 FP16 TFLOPS | 130 INT8 TOPS | 260 INT4 TOPS
16GB | 320GB/s
70 W

Deep Learning Training & Inference
HPC Workloads
Video Transcode
Remote Graphics

TensorRT 5 & TensorRT inference server

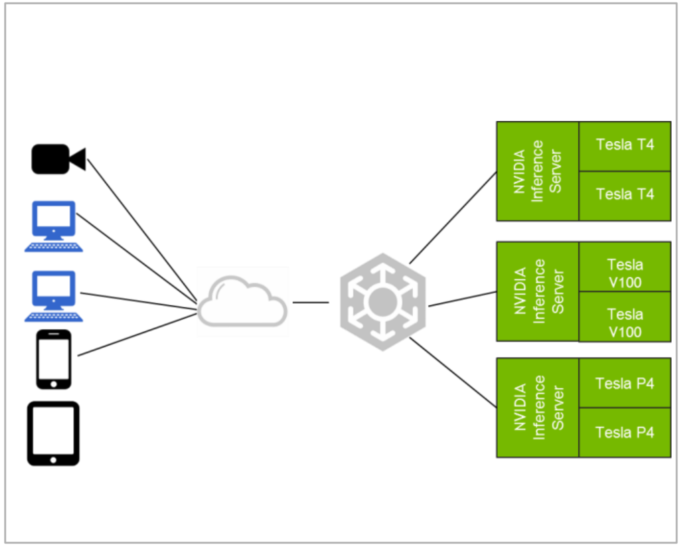
Turing Support • Optimizations & APIs • Inference Server

World's Most Advanced Inference Accelerator

Up to 40x faster perf. on Turing Tensor Cores

New optimizations & flexible INT8 APIs

New INT8 workflows, Win & CentOS support



TensorRT inference server

Maximize GPU utilization, run multiple models on a node

Free download to members of NVIDIA Developer Program at

developer.nvidia.com/tensorrt

T4/TRT5 Inference Results on U-net

TF-TRT for fast prototyping, TRT for maximum performance

23.5x speed-up by native TRT (INT 8 precision)

Inference method		CPU-TF	GPU-TF	TF-TRT5	TRT5
FP 32 bit	images/sec	38.6	230.4	320.0	438.8
	perf. Increase	1	5.8	8.1	11.1
FP 16 bit	images/sec	N/A	N/A	334.0	501.0
	perf. Increase	N/A	N/A	8.4	12.6
INT 8 bit	images/sec	N/A	N/A	459.0	909.0
	perf. Increase	N/A	N/A	11.9	23.5

SUMMARY

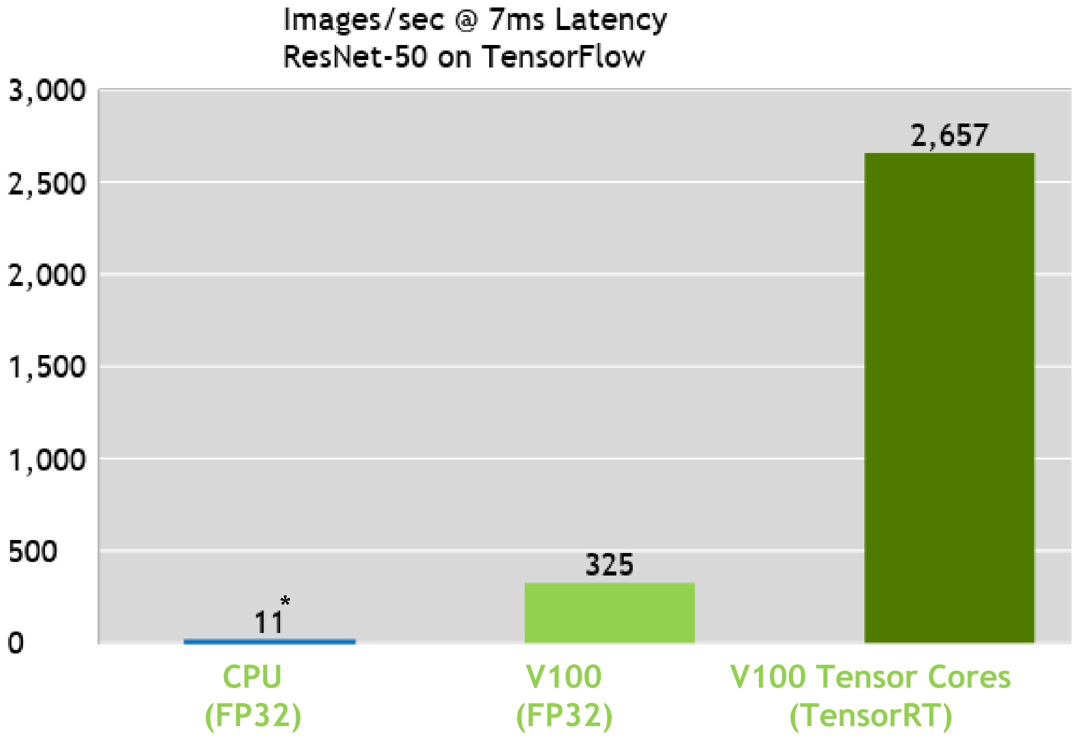
Challenges	Delivers
Training , inference environment is hard to build, maintain, share.	Using NGC Docker images.
Model optimizations and speed up throughput.	TF-TRT or TensorRT
So many deep learning model out there, how to choose the right model?	If your dataset, demand requirement fit the scenario like we do. U-Net model is great choice for segmentation task.
Inference Service Architect hard to develop	NGC ready TRTIS and open sourced, easy set up

Thank
You

Appendix

TensorRT INTEGRATED WITH TensorFlow

TRT4: Delivers 8x Faster Inference



* Min CPU latency measured was 83 ms. It is not < 7 ms.

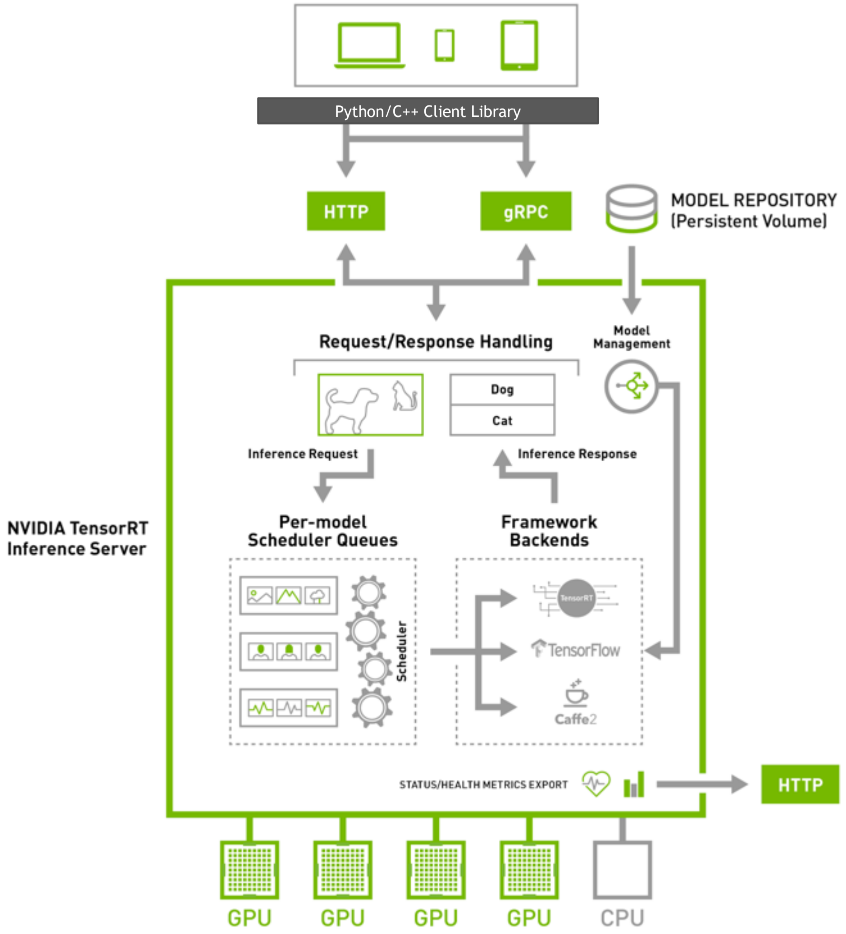
- AI Researchers
- Data Scientists

Available in TensorFlow 1.7+
https://github.com/tensorflow/tensorflow

CPU: Skylake Gold 6140, 2.5GHz, Ubuntu 16.04; 18 CPU threads.
Volta V100 SXM; CUDA (384.111; v9.0.176);
Batch size: CPU=1, TF_GPU=2, TF-TRT=16 w/ latency=6ms

INFERENCE SERVER ARCHITECTURE

Available with Monthly Updates



Models supported

- TensorFlow GraphDef/SavedModel
- TensorFlow and TensorRT GraphDef
- TensorRT Plans
- Caffe2 NetDef (ONNX import)

Multi-GPU support

Concurrent model execution

Server HTTP REST API/gRPC

Python/C++ client libraries

TESLA PRODUCT FAMILY

TESLA V100 (Scale-up)

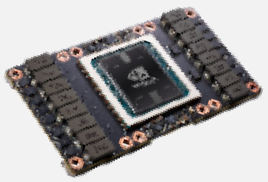
Supercomputing
DL Training & Inference
Machine Learning
Video | Graphics

HGX-2 Baseboard

16 V100 + NVSwitch

V100 SXM2

with NVLINK



V100 PCIe

2 slot

TESLA T4 (Scale-out)

DL Inference &
Training
Machine Learning
Video | Graphics

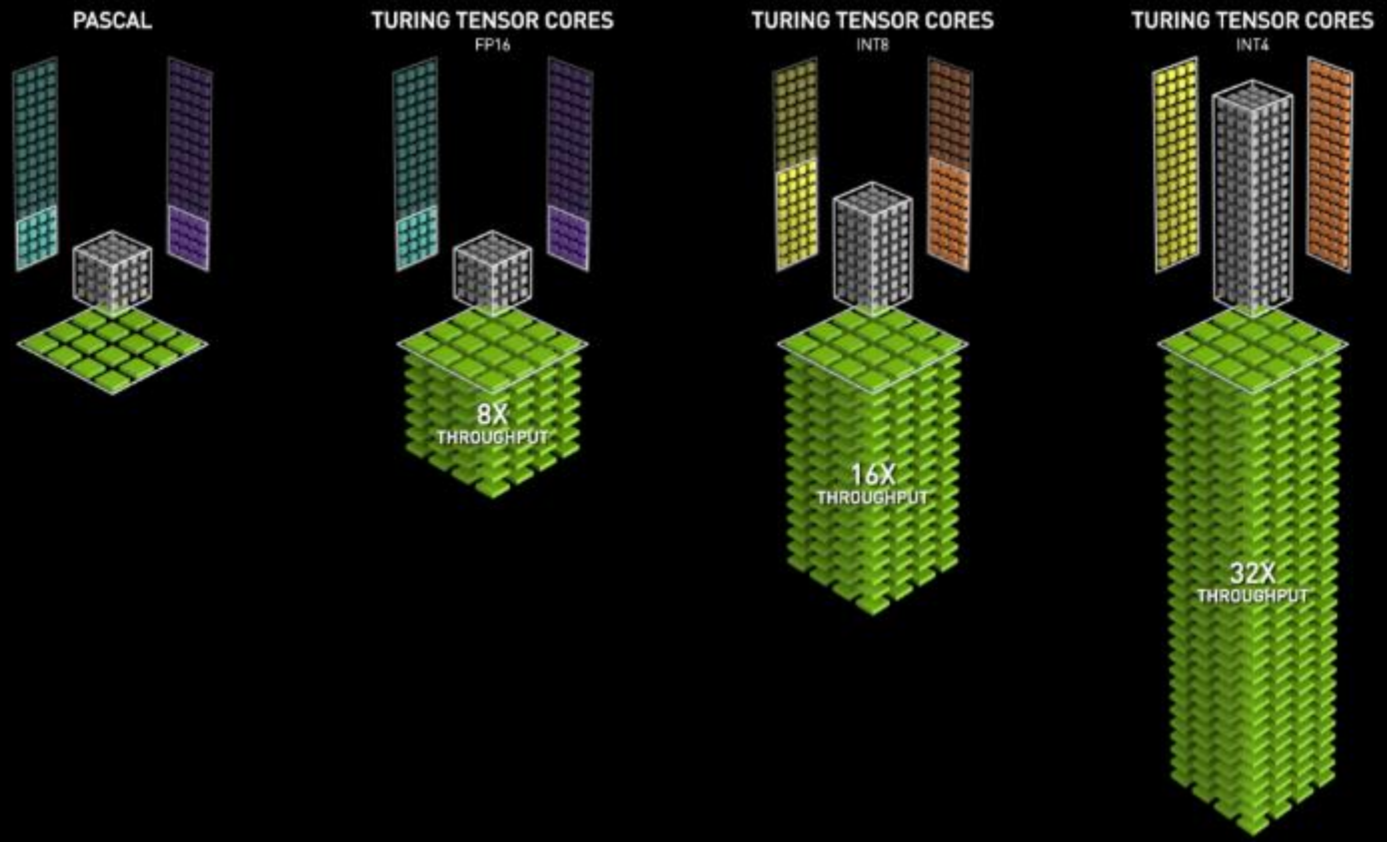
T4 PCIe

Low Profile

HGX-2: V100 & NVSwitch heat sink included but not shown

NEW TURING TENSOR CORE

MULTI-PRECISION FOR AI INFERENCE & SCALE-OUT TRAINING
65 TFLOPS FP16 | 130 TeraOPS INT8 | 260 TeraOPS INT4



TensorRT 5 Supports Turing GPUs

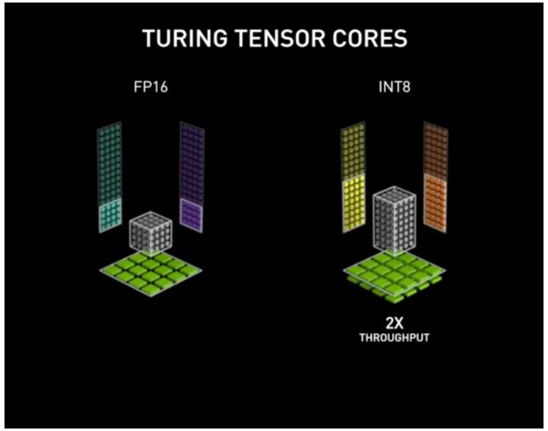
Fastest Inference Using Mixed Precision (FP32, FP16, INT8) and Turing Tensor Cores

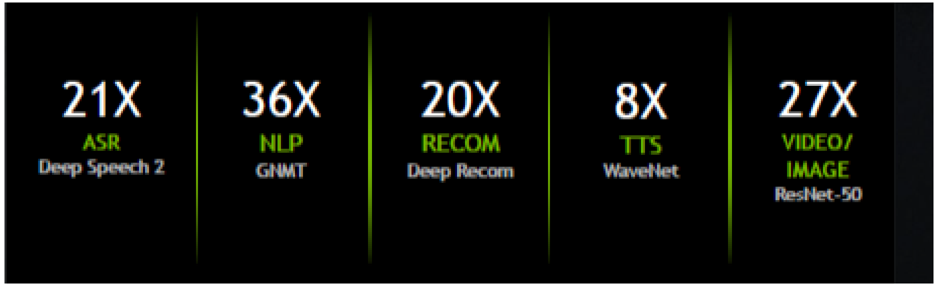
Speed up recommender, speech, video and translation in production

Optimized kernels for mixed precision (FP32, FP16, INT8) workloads on Turing GPUs

Up to 40x faster inference for apps vs CPU-only platforms

MPS maximizes utilization with multiple separate inference processes





developer.nvidia.com/tensorrt